Field of the Invention

The present invention relates to computers and, in particular, to a modified machine architecture which enables improved performance to be achieved.

Background Art

Ever since the advent of computers, and computing, software for computers has been written to be operated upon a single machine. As indicated in Fig. 1, that single prior art machine 1 is made up from a central processing unit, or CPU, 2 which is connected to a memory 3 via a bus 4. Also connected to the bus 4 are various other functional units of the single machine 1 such as a screen 5, keyboard 6 and mouse 7.

A fundamental limit to the performance of the machine 1 is that the data to be manipulated by the CPU 2, and the results of those manipulations, must be moved by the bus 4. The bus 4 suffers from a number of problems including so called bus "queues" formed by units wishing to gain an access to the bus, contention problems, and the like. These problems can, to some extent, be alleviated by various stratagems including cache memory, however, such stratagems invariably increase the administrative overhead of the machine 1.

Naturally, over the years various attempts have been made to increase machine performance. One approach is to use symmetric multiple processors. This prior art approach has been used in so called "super" computers and is schematically indicated in Fig. 2. Here a plurality of CPU's 12 are connected to global memory 13. Again, a bottleneck arises in the communications between the CPU's 12 and the memory 13. This process has been termed "Single System Image". There is only one application and one whole copy of the memory for the application which is distributed over the global memory. The single application can read from and write to, (ie share) any memory location completely transparently.

Where there are a number of such machines interconnected via a network, this is achieved by taking the single application written for a single machine and partitioning the required memory resources into parts. These parts are then distributed across a number of computers to form the global memory 13 accessible by all CPU's 12. This procedure relies on masking, or hiding, the memory partition from the single running application program. The performance degrades when one CPU on one

machine must access (via a network) a memory location physically located in a different machine.

Although super computers have been technically successful in achieving high computational rates, they are not commercially successful in that their inherent complexity makes them extremely expensive not only to manufacture but to administer. In particular, the single system image concept has never been able to scale over "commodity" (or mass produced) computers and networks. In particular, the Single System Image concept has only found practical application on very fast (and hence very expensive) computers interconnected by very fast (and similarly expensive) networks.

A further possibility of increased computer power through the use of a plural number of machines arises from the prior art concept of distributed computing which is schematically illustrated in Fig. 3. In this known arrangement, a single application program (Ap) is partitioned by its author (or another programmer who has become familiar with the application program) into various discrete tasks so as to run upon, say, three machines in which case n in Fig. 3 is the integer 3. The intention here is that each of the machines M1...M3 runs a different third of the entire application and the intention is that the loads applied to the various machines be approximately equal. The machines communicate via a network 14 which can be provided in various forms such as a communications link, the internet, intranets, local area networks, and the like. Typically the speed of operation of such networks 14 is an order of magnitude slower than the speed of operation of the bus 4 in each of the individual machines M1, M2, etc.

Distributed computing suffers from a number of disadvantages. Firstly, it is a difficult job to partition the application and this must be done manually. Secondly, communicating data, partial results, results and the like over the network 14 is an administrative overhead. Thirdly, the need for partitioning makes it extremely difficult to scale upwardly by utilising more machines since the application having been partitioned into, say three, does not run well upon four machines. Fourthly, in the event that one of the machines should become disabled, the overall performance of the entire system is substantially degraded.

A further prior art arrangement is known as network computing via "clusters" as is schematically illustrated in Fig. 4. In this approach, the entire application is loaded onto each of the machines M1, M2 ....Mn. Each machine communicates with a common database but does not communicate directly with the other machines. Although each machine runs the same application, each machine is doing a different "job" and uses only its own memory. This is somewhat analogous to a number of windows each of which sell train tickets to the public. This approach does operate, is scalable and mainly suffers from the disadvantage that it is difficult to administer the network.

Object of the Invention

The object of the present invention is to provide a modified machine architecture which goes some way towards overcoming, or at least ameliorating, some of the abovementioned disadvantages.

Summary of the Invention

In accordance with a first aspect of the present invention there is disclosed a plurality of computers interconnected via a communications link and operating at least one application program simultaneously.

In accordance with a second aspect of the present invention there is disclosed a method of loading an application program onto each of a plurality of computers, the computers being interconnected via a communications link, the method comprising the step of modifying the application as it is being loaded.

In accordance with a third aspect of the present invention there is disclosed a method of operating at least one application program simultaneously on a plurality of computers all interconnected via a communications link and each having at least a minimum predetermined local memory capacity, said method comprising the steps of:

(i) initially providing each local memory in substantially identical condition,

(ii) satisfying all memory reads and writes generated by said application program from said local memory, and

(iii) communicating via said communications link all said memory writes at each said computer which take place locally to all the remainder of said plurality of

computers whereby the contents of the local memory utilised by each said computer subject to an updating data transmission delay, remains substantially identical.

In accordance with a fourth aspect of the present invention there is disclosed a method of compiling or modifying an application program to run simultaneously on a plurality of computers interconnected via a communications link, said method comprising the steps of:

(i) detecting instructions which share memory records

(ii) listing all such shared memory records and providing a naming tag for each listed memory record

(iii) detecting those instructions which write to, or manipulate the contexts of, any of said listed memory records, and

(iv) generating an alert instruction following each said detected write or manipulate instruction, said alert instruction forwarding the re-written or manipulated contents and name tag of each said re-written or manipulated listed memory record.

In accordance with a fifth aspect of the present invention there is disclosed in a multiple thread processing computer operation in which individual threads of a single application program are simultaneously being processed each on a corresponding one of a plurality of computers interconnected via a communications link, the improvement comprising communicating changes in the contents of local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link.

Brief Description of the Drawings

Embodiments of the present invention will now be described with reference to the drawings in which:

Fig. 1 is a schematic view of the internal architecture of a conventional computer,

Fig. 2 is a schematic illustration showing the internal architecture of known symmetric multiple processors,

Fig. 3 is a schematic representation of prior art distributed computing,

Fig. 4 is a schematic representation of a prior art network computing using clusters,

Fig. 5 is a schematic block diagram of a plurality of machines operating the same application program in accordance with a first embodiment of the present invention,

Fig. 6 is a schematic illustration of a prior art computer arranged to operate JAVA code and thereby constitute a JAVA virtual machine,

Fig. 7 is a drawing similar to Fig. 6 but illustrating the initial loading of code in accordance with the preferred embodiment,

Fig. 8 is a drawing similar to Fig. 5 but illustrating the interconnection of a plurality of computers each operating JAVA code in the manner illustrated in Fig. 7,

Fig. 9 is a flow chart of the procedure followed during loading of the same application on each machine in the network,

Fig. 10 is a schematic representation of multiple thread processing carried out on the machines of Fig. 8 utilizing a first embodiment of memory updating,

Fig. 11 is a schematic representation similar to Fig. 10 but illustrating an alternative embodiment,

Fig. 12 is a schematic representation of two laptop computers interconnected to simultaneously run a plurality of applications, with both applications running on a single computer,

Fig. 13 is a view similar to Fig. 12 but showing the Fig. 12 apparatus with one application operating on each computer, and

Fig. 14 is a view similar to Figs. 12 and 13 but showing the Fig. 12 apparatus with both applications operating simultaneously on both computers.

Detailed Description

In connection with Fig. 5, in accordance with a preferred embodiment of the present invention a single application can be operated simultaneously on a number of machines M1, M2 .....Mn. As it will become apparent hereafter, each of the machines operates with the same code and data on each machine and thus all of the machines have the same code and data. In addition, the code stored on each machine has been modified by the same rules (or substantially the same rules since minor optimising changes are permitted).

As a consequence of the above described arrangement, if each of the machines M1, M2 ....Mn has, say, a shared memory capability of 10MB, then the total shared memory available to the application is not, as one might expect 10n MB but rather only 10MB. However, this results in improved operation will become apparent hereafter. Naturally, each machine has an unshared memory capability. The unshared memory capability of the machines are normally approximately equal but need not be.

It is known from the prior art to operate a machine (produced by one of various manufacturers and having an operating system operating in one of various different languages) in a particular language of the application, by creating a virtual machine. Thus, where the intended language of the application is the language JAVA, a JAVA virtual machine is created which is able to operate code in JAVA irrespective of the machine manufacturer and internal details of the machine.

This well known prior art arrangement is modified in accordance with the preferred embodiment of the present invention by the provision of an additional facility which is conveniently termed "distributed run time" or DRT. In particular, the distributed run time comes into operation during the loading of the JAVA code so as to initially create the JAVA virtual machine. The sequence of operation during loading will be described hereafter in relation to Fig. 9.

Fig. 8 shows in modified form the arrangement of Fig. 5 utilising JAVA virtual machines as illustrated in Fig. 7. It will be apparent that again the same application code and data are loaded onto each machine, however, the communications between each machine although physically routed through the machine hardware, are controlled by the individual DRT within each machine. Thus, in practice this may be conceptionalised as the DRT's communicating with each other via the network rather than the machines themselves.

Turning now to Figs. 7 and 9, during the loading procedure, the program being loaded to create the JAVA virtual machine is modified. This modification involves the initial step of detecting all JAVA fields (or equivalent in other languages) in the application being loaded. Such fields share memory and thus need to be identified for subsequent processing. The DRT during the loading procedure creates a list of all the

fields thus identified, the fields being listed by object and class. Both volatile and synchronous fields are listed.

The next phase of the modification procedure is to search through the executable application code in order to locate every processing activity that manipulates field values or writes to field values. When such an operation (typically put static or put field) is detected which changes the field, then through the "instrument" instruction the byte code at that point in the program is changed to insert an alert to the DRT that the value of the field has changed. Thereafter, the loading program continues in a normal way.

Once this initial modification during the loading procedure has taken place, then either one of the multiple thread processing operations illustrated in Figs. 10 and 11 takes place. As seen in Fig. 10, multiple thread processing on the machines is occurring and the processing of the second thread (in this example) results in the DRT of that thread being alerted to a change of field value. At this stage the processing of that thread is halted, and the same thread notifies all other DRTs via the network of the changed value. At the end of that communication procedure, the thread then resumes the processing until the next instance where the DRT is alerted to a change of field value.

In the alternative arrangement illustrated in Fig. 11, once the DRT of a thread has been alerted to a change of field value, it instructs the DRT of another thread to notify all other DRTs of the changed value. This is an operation which can be carried out quickly and thus the processing of the "altered" thread is only interrupted momentarily before the thread resumes processing. The other thread which has been notified of the change then communicates that change to each of the other machines. This embodiment makes better utilisation of the processing power of the various threads (which are not, in general, subject to equal demands) and gives better scaling with increasing size of "n", being an integer greater than or equal to 2 which represents the total number of machines connected to the network. Irrespective of which embodiment is used, the changed field values are propagated to all the other machines on the network.

In the prior art arrangement utilising distributed software, memory accesses from one machine's software to memory physically located on another machine are permitted by the network interconnecting the machines. However, such memory accesses can result in delays in processing of the order of $10^6 - 10^7$ cycles of the central processing unit of the machine. This in large part accounts for the diminished performance of the multiple interconnected machines.

However, in the present arrangement as described above, it will be appreciated that all reading of data is satisfied locally because the current value of all fields is stored on the machine carrying out the processing which generates the demand to read memory. Such local processing can be satisfied within $10^2 - 10^3$ cycles of the central processing unit. Thus, in practice, there is substantially no waiting for memory accesses which involves reads.

However, most application software reads memory frequently but writes to memory relatively infrequently. As a consequence, the rate at which memory is being written or re-written is relatively slow compared to the rate at which memory is being read. Because of this slow demand for writing or re-writing of memory, the fields can be continually updated at a relatively low speed via the inexpensive commodity network, yet this low speed is sufficient to meet the application program's demand for writing to memory.

In a further modification in relation to the above, the changes to fields can be grouped into batches so as to further reduce the demands on the communication speed of the network interconnecting the various machines.

It will also be apparent to those skilled in the art that in the table created by each DRT when initially recording the fields, for each field there is a name which is common throughout the network and which the network recognises. However, in the individual machines the memory location corresponding to a given name field will vary over time since each machine will progressively store changed field values at different locations according to its own internal processes. Thus the table in each of the DRTs will have, in general, different memory locations but each "field name" will have the same "field value" stored in the different memory locations.

It will also be apparent to those skilled in the art that the abovementioned modification of the application program during loading can be accomplished in up to four ways by re-compilation at loading, by a pre-compilation procedure prior to loading, by a "just-in-time" compilation, or by re-compilation after loading. Traditionally the term "compilation" implies a change in code or language, eg from source to object code or one language to another. Clearly the use of the term "compilation" (and its grammatical equivalents) in the present specification is not so restricted and can embrace modifications within the same code or language.

Turning now to Figs. 12-14, two laptop computers 101 and 102 are illustrated. The computers 101 and 102 are not necessarily identical and indeed, one can be an IBM-clone and the other can be an APPLE computer. The computers 101 and 102 have two screens 105, 115 two keyboards 106, 116 but a single mouse 107. The two machines 101, 102 are interconnected by a means of a single coaxial/twisted pair ?? cable 114.

Two simple application programs are downloaded onto each of the machines 101, 102, the programs being modified as they are being loaded as described above. In this embodiment the first application is a simple calculator program and results in the image of a calculator 108 being displayed on the screen 105. The second program is a graphics program which displays four coloured blocks 109 which are of different colours and which move about at random within a rectangular box 110. Again, after loading, the box 110 is displayed on the screen 105. Each application operates independently so that the blocks 109 are in random motion on the screen 105 whilst numerals within the calculator 108 can be selected (with the mouse 107) together with a mathematical operator (such as additional multiplication) so that the calculator 108 displays the result.

The mouse 107 can be used to "grab" the box 110 and move same to the right across the screen 105 and onto the screen 115 so as to arrive at the situation illustrated in Fig. 13. In this arrangement, the calculator application is being conducted on machine 101 whilst the graphics application resulting in display of box 110 is being conducted on machine 102.

However, as illustrated in Fig. 14, it is possible by means of the mouse 107 to drag the calculator 108 to the right as seen in Fig. 13 so as to have a part of the calculator 108 displayed by each of the screens 105, 115. Similarly, the box 110 can be dragged by means of the mouse 107 to the left as seen in Fig. 13 so that the box 110 is partially displayed by each of the screens 105, 115 as indicated Fig. 14. In this configuration, part of the calculator operation is being performed on machine 101 and part on machine 102 whilst part of the graphics application is being carried out the machine 101 and the remainder is carried out on machine 102.

The foregoing describes only some embodiments of the present invention and modifications, obvious to those skilled in the art, can be made thereto without departing from the scope of the present invention.

The term "comprising" (and its grammatical variations) as used herein is used in the inclusive sense of "having" or "including" and not in the exclusive sense of "consisting only of".